(4)

Multi-Level Specification and Verification of Real-Time Software*

Armen Gabrielian and Matthew K. Franklin

Technical Report 89-14

July 1989

THOMSON-CSF, INC.
PACIFIC RIM OPERATIONS

630 Hansen Way, Suite 250
Palo Alto, California 94304

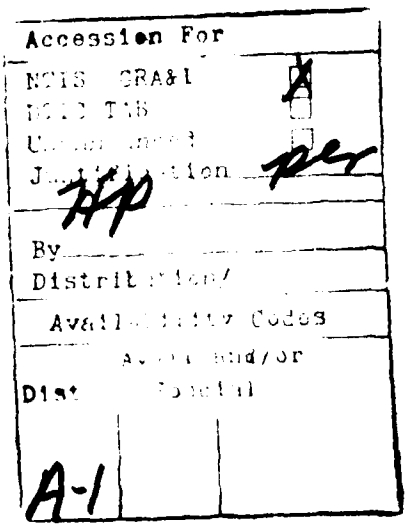
4

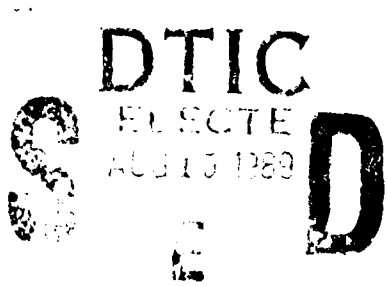
Multi-Level Specification and Verification of Real-Time Software*

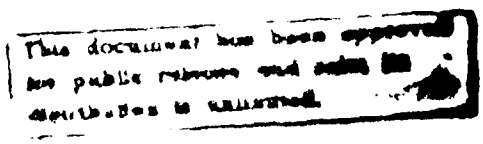
Armen Gabrielian and Matthew K. Franklin

Technical Report 89-14

July 1989







* The work reported in this paper was supported in part by the Office of Naval Research under Contract No. N00014-89-C-0022.

Multi-Level Specification and Verification of Real-Time Software*

Armen Gabrielian and Matthew K. Franklin

Thomson-CSF, Inc.
630 Hansen Way, Suite 250
Palo Alto, CA 94304

Abstract

A state-based approach is presented for specification of real-time software at multiple levels of abstraction. In this approach, specification at each level is performed in terms of a Hierarchical Multi-State (HMS) machine, with the higher levels defining requirements and the lower levels indicating methods of achieving the requirements. This leads to a considerable simplification of the specification process, can lead to reusability of specifications and is fundamentally different from the refinement approach to specification. A restricted method of verifying the consistency of multi-level specifications is also presented. By the use of this method, necessary scheduling of concurrent tasks to satisfy a complex set of logical and temporal constraints can be derived.

Keywords: Formal specification, verification, state models, real-time systems, requirements specification, real-time scheduling, automata, HMS machines.

1. Introduction

Three critical problems in the early phases of the life-cycle of real-time software are: (1) definition of requirements, (2) specification of the system at a conceptual level prior to software design, and (3) verification that the requirements are satisfied by the specification. The objectives of this paper are to present a state-based specification method for real-time software, in which,

- Specification is performed in terms of *nondeterministic* abstract machines that represent the generic behavior of entire classes of software.
- Requirements are stated as higher-level abstract machines that define the desirable behaviors of the nondeterministic lower-level specifications.
- Formal techniques can be applied to verify whether a specification can be *executed* in such a manner that the requirements are satisfied.

The notion of multi-level specification presented in this paper is fundamentally different from the "refinement" approach to hierarchical specification. In the latter approach, typified by [Ab-88] and common to the computer security field (see, e.g., [Ch-81]), one usually begins with a top-level specification. Thereafter, one creates successively lower-level independent specifications that are proven to be consistent with higher levels. At the end of this process, the lowest level stands alone as the final specification. In contrast, in our approach, each higher-level specification imposes constraints on lower-level specifications such that all levels remain part of the final specification. Moreover, each level can be reused in other contexts.

---

* The work reported in this paper was supported in part by the Office of Naval Research under Contract No. N00014-89-C-0022.

Numerous techniques have been proposed in the literature for specification of real-time software (see, e.g., some of the papers in [Ge-86] and [St-88]). The approach presented here is an extension of the Hierarchical Multi-State (HMS) machine specification method presented in [Ga-88a], [Ga-88b] and [Fr-89]. An earlier version of HMS machines was introduced in [Ga-87] for causal modeling of dynamic systems. Intuitively, HMS machines are high-level automata, in which multiple states can be active at any moment of time, multiple transitions can occur simultaneously, states can be decomposed hierarchically into lower-level machines, and there exists a rich language for representing logical and temporal dependencies among states and transitions.

The benefits of using the HMS machine model in [Ga-88a] for the specification of real-time systems are (1) orders of magnitude reduction in the number of states compared to finite-state machines and similar formalisms, (2) ability to represent and reason about concurrency and hard real-time constraints, (3) executability, and (4) formal techniques for verifying that "safety properties" are not violated [Fr-89].

The HMS machine model has some similarities and major differences with a number of other state-based modeling techniques in the literature such as Petri nets [Re-82], Statecharts [Ha-87] and Modecharts [Ja-88]. The most important similarity is that multiple states may be true or "marked" at any moment of time, thereby reducing the complexity of the state space. In comparison with Petri nets, HMS machines provide a higher-level language that avoids the need for intermediate dummy states that are often needed to maintain logical consistency in Petri nets. Also, while the semantics of Petri nets makes it difficult to distinguish between *precedence* and *causality,* causal relationships are easily represented in HMS machines [Ga-87]. In comparison with Statecharts and Modecharts, the major differences in the basic models are in the form of controls on transitions, the language of time, the rules of execution, and the graphic notation.

The purpose of this paper is two-fold: (1) We introduce an approach for constraining nondeterministic HMS specifications in terms of higher-level HMS "policy machines" that both define requirements and suggest how requirements can be achieved. For a software system, a lower-level machine can represent the specification of a "generic" program, where, besides the types of variables, other features may have been left undefined. This replaces and extends the prioritized policy method of [Ga-88a]. (2) We present a new method for verifying that certain conditions with finite temporal bounds are achievable in a specification. Specifically, we demonstrate that if a *choice* of actions (such as calls to software modules) to reach a goal state is known, then one can derive analytically the precise *ordering* and *intermediate delays* between the actions that are necessary to meet a complex set of logical and temporal constraints. This method is complementary to the verification methods of [Fr-89], where correctness-preserving transformations were employed to demonstrate that states corresponding to undesirable safety conditions are *unreachable.*

Section 2 of this paper introduces the definition of basic HMS machines, along with examples of the graphic notation for representing them. Section 3 presents our approach to the

use of higher-level HMS machines to constrain lower-level nondeterministic HMS machines. Section 4 introduces our method of verification of HMS specifications and Section 5 presents an example demonstrating the utility of the multi-level specification and verification methods. The summary is presented in Section 6.

2. Definition of HMS Machines

In this section, we present the basic definitions for "HMS machines," which constitute our state-based specification formalism at the lowest level of abstraction. In the next section, we extend slightly the definitions and present an approach to "multi-level" specification, where HMS machines at different levels of abstraction interact with each other.

We begin by introducing a language for representing temporal constraints. Then, we define how a machine is constructed in terms of its states and transitions (which are controlled by other states). Finally, we present formally how changes occur in a machine as time progresses.

Definition 1 (Time Expression): τ is a "time expression" if τ is of the form $<t_1, t_2>$, $<t_1, t_2>!$, or $[t_1, t_2]$, where t_1 and t_2 are integers and $t_1 \leq t_2 \leq 0$. When $t_1 = t_2 = t$, these three time expressions will be written as t, t!, and t, respectively.

The time expressions have informal names: (1) $<t_1, t_2>$ is called a "sometime-delay" and represents existential quantification of time over the period from t_1 to t_2, (2) $[t_1, t_2]$ is called an "always-delay" and represents universal quantification of time for the period from t_1 to t_2, and (3)$<t_1, t_2>!$ is called a "sometime-change-delay." The precise interpretation of the different time expressions will be presented in Definition 6.

Definition 2 (Control): c is a "control" over a set of "states" S if τ is a time expression and c is (s, τ) or $(\neg s, \tau)$, for some state $s \in S$.

Definition 3 (Transition): γ is a "transition" over the set of states S if γ is of the form

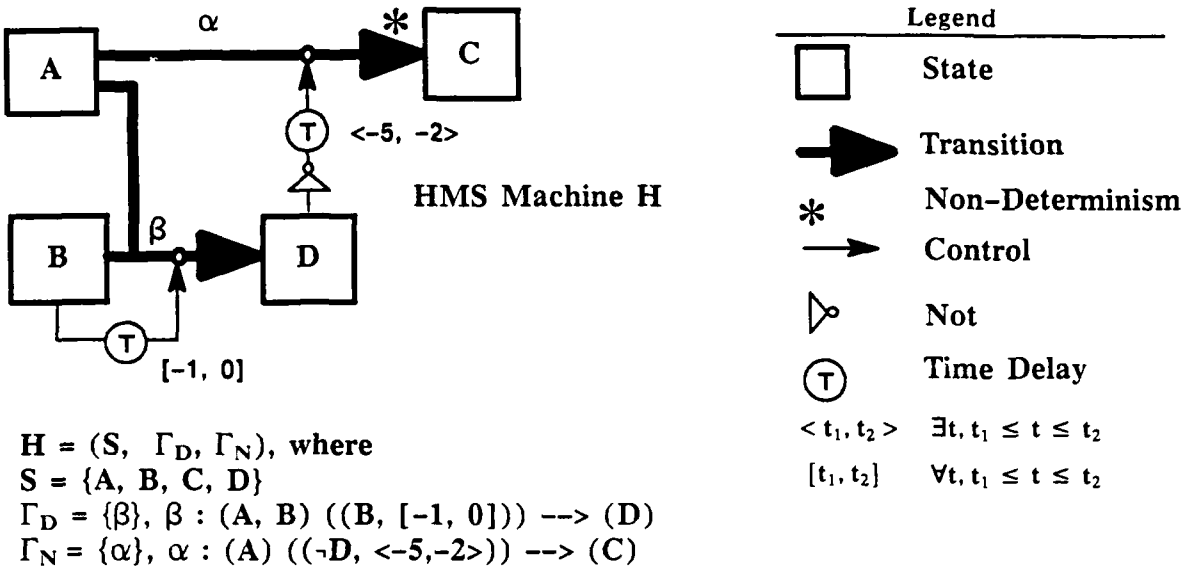
id: (PRIMARIES) (CONTROLS) --> (CONSEQUENTS)

where id is a "label," PRIMARIES is a (possibly empty) subset of S, CONTROLS is a (possibly empty) set of controls over S, and CONSEQUENTS is a (possibly empty) subset of S. These three sets will be denoted PRIMS(γ), CTRLS(γ) and CNSQS(γ), respectively.

We will now introduce the simplest version of a Hierarchical Multi-State (HMS) machine.

Definition 4 (Basic HMS Machine): A "basic HMS machine" is a triple $H = (S, \Gamma_D, \Gamma_N)$, where S is a set of states, and Γ_D and Γ_N are (possibly empty) sets of transitions over S. Γ_D and Γ_N will be called the "deterministic" and "nondeterministic" transitions of H, respectively. When there is no confusion, a basic HMS machine will simply be called an "HMS machine."

Figure 2.1 presents our graphic notation for representing HMS machines. As indicated by the legend in the figure, boxes denote states, dark arrows show transitions, and thin arrows represent controls, with VLSI notation defining logical operations on controls.



H = (S, Γ_D, Γ_N), where
S = {A, B, C, D}
Γ_D = {β}, β : (A, B) ((B, [-1, 0])) --> (D)
Γ_N = {α}, α : (A) ((¬D, <-5,-2>)) --> (C)

Figure 2.1. Graphic Notation for HMS Machine

We now turn to the definition of the "execution" of an HMS machine. Informally, a machine executes by "firing" some of its transitions at each moment, thereby altering the values or "markings" of some or all of its states at the next moment. A generalized concept of marking will now be introduced that allows the representation of the history of an HMS machine behavior for the infinite past. The assumption of an infinite past will simplify later definitions, but only finite histories are needed in practice.

Definition 5 (Marking): M is a "marking" of an HMS machine H = (S, Γ_D, Γ_N) if M is a mapping from S × {0, -1, -2, ...} to {T, F}. If M(s, i) = T (F), then the state s is said to be "marked" ("unmarked") or "true" ("false") at time i. All times are relative to the current moment, which is denoted by 0.

Given an HMS machine, at any moment of time, some of its transitions are "enabled." An HMS machine executes by "firing" all its enabled deterministic transitions and a subset (possibly empty) of its enabled nondeterministic transitions, thereby resulting in a new marking for the *next* moment. This process is formalized by Definitions 6-10:

Definition 6 (Control Satisfaction): The control c is "satisfied" by a marking M if
(i) c is (s, <t_1, t_2>) and M(s, t_3) = T for some t_3 s.t. $t_1 \leq t_3 \leq t_2$.
(ii) c is (s, [t_1, t_2]) and M(s, t_3) = T for every t_3 s.t. $t_1 \leq t_3 \leq t_2$.
(iii) c is (s, <t_1, t_2>!), M(s, t_3) = T for some t_3 s.t. $t_1 \leq t_3 \leq t_2$, and M(s, t_1 - 1) = F.
(iv) c is (¬s, <t_1, t_2>) and M(s, t_3) = F for some t_3 s.t. $t_1 \leq t_3 \leq t_2$.
(v) c is (¬s, [t_1, t_2]) and M(s, t_3) = F for every t_3 s.t. $t_1 \leq t_3 \leq t_2$.
(vi) c is (¬s, <t_1, t_2>!), M(s, t_3) = F for some t_3 s.t. $t_1 \leq t_3 \leq t_2$, and M(s, t_1 - 1) = T.

Given a set of controls C, we write M ⊢ C, if every member of C is satisfied by M.

Definition 7 (Transition Enablement): The transition γ is "enabled" for marking M if (1) M(s, 0) = T for all s in PRIMS(γ), and (2) c is satisfied by M for all c in CTRLS(γ).

For convenience, we define the following sets for $H = (S, \Gamma_D, \Gamma_N)$ and marking M:

D-ENAB(H, M) = $\{\gamma \mid \gamma \in \Gamma_D$ and γ enabled for M$\}$,
N-ENAB(H, M) = $\{\gamma \mid \gamma \in \Gamma_N$ and γ enabled for M$\}$.

Definition 8 (Firing Set): The set of transitions Γ is a "firing set" of $H = (S, \Gamma_D, \Gamma_N)$ for marking M if Γ = D-ENAB(H, M) $\cup\ \Gamma'$, where $\Gamma' \subseteq$ N-ENAB(H, M).

Definition 9 (Next Marking): If M is a marking of an HMS machine H, and if Γ is a firing set of H for M, then the marking "after M via Γ" is denoted by M[Γ], and is given by

M[Γ](s, 0) = T for every state s in CNSQS(Γ)
M[Γ](s, 0) = F for every state s in PRIMS(Γ) but not in CNSQS(Γ)
M[Γ](s, 0) = M(s, 0) for every state s not in either PRIMS(Γ) or CNSQS(Γ)
M[Γ](s, t) = M(s, t + 1) for every $s \in S$, and every time $t < 0$.

Definition 10 (HMS Execution): If H is an HMS machine, and if M_0 is a marking of H, then an "execution of H from M_0" is a sequence $[M_0, M_1, M_2, \ldots]$ of markings such that

$M_{i+1} = M_i[\Gamma_i]$ for some firing set Γ_i of H for M_i, for each $i \geq 0$.

The set of all executions of H from M_0 is denoted by $\mathcal{E}(H, M_0)$. Intuitively, at each moment, all the enabled deterministic transitions and a subset of the enabled nondeterministic transitions fire, thereby causing changes in the marking of H.

An HMS machine can be considered as a "specification" of the dynamic behavior of a system. For example, Figure 2.1 depicts the execution of a simple HMS machine with four states and only deterministic transitions. Initially, only the state A is marked (denoted by small dark circle). In the subsequent three time steps certain transitions fire causing other states to be marked and some marked states to be unmarked. For example, at time t+2, the transition α is enabled since (1) its primary state A is marked and (2) the control (B, <-2, -1>) is satisfied since the state B was marked at time (t+2)-1.

The "hierarchical" version of HMS machines (the "H" in HMS) is obtained by allowing each state to be an HMS machine itself and designating certain states as "initial states" and "final states." A definition of this was presented in [Ga-88a], along with a discussion of the *structural constraints* and *execution protocols* that are necessary to execute such machines. In this paper, the hierarchical version of HMS machines will not be considered. Instead, we turn next to the consideration of the much more powerful notion of higher-level HMS machines that control lower-level HMS machines.

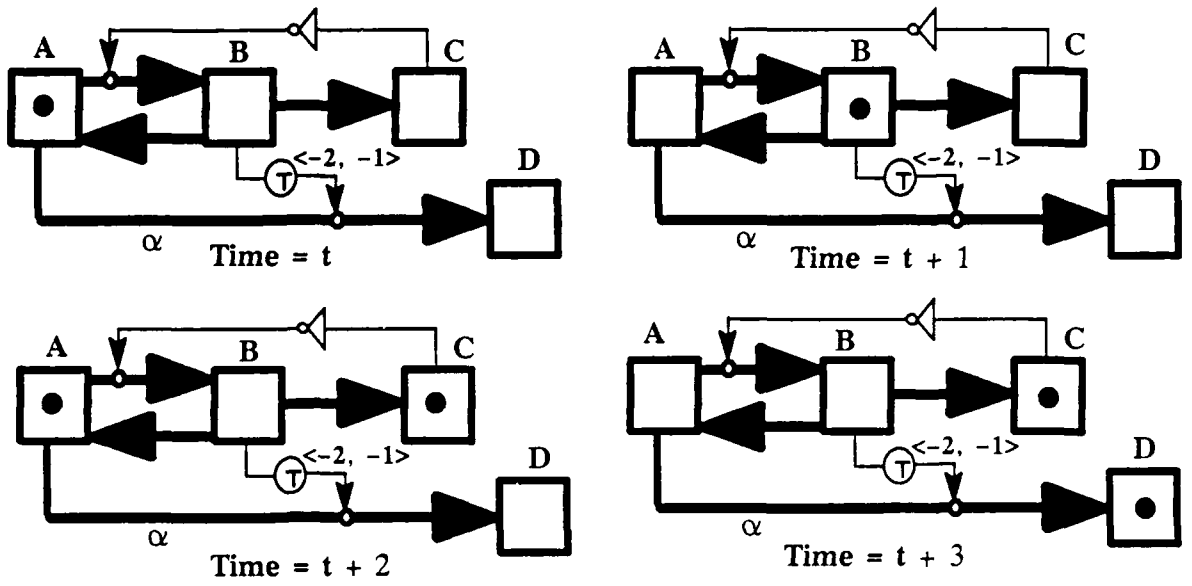


Figure 2.2. Example of an HMS Machine Execution

3. Policy Machines and Requirements

An HMS machine with nondeterministic transitions can be considered as an executable specification of the *generic* structure of a software system that satisfies a general set of requirements. It can serve as a reusable specification that is particularized by the addition of further constraints. In [Ga-88a], these further constraints were provided as a set of "policies" of the form

[priority] precondition ---> postcondition,

where each precondition and postcondition is a predicate on the markings of states. A policy becomes "active" if it is the highest-priority policy for which its precondition is true. An active policy then is "triggered" by finding a path through the nondeterministic transitions of the associated HMS machine that makes its postcondition true. The execution of the HMS machine along this path then causes the satisfaction of the policy. In [Ga-88a], an example of a 12-state HMS machine with six policies was demonstrated that provides a complete specification of an elevator for any number of floors. Without the use of policies, the specification would have been considerably more complicated since it would have been necessary to specify the desirable paths explicitly.

A more powerful means of particularizing an HMS machine with nondeterministic transitions is by controlling its execution with other HMS machines. We introduce a new type of HMS machine that can define the requirements (or constraints) on the execution of a lower-level machine. An ordered list of machines, with the lowest one being a basic HMS machine, will then constitute a multi-level specification of a software system. This improves on the two-level method of policies in three important respects: (1) specification can be

performed at arbitrary number of levels; (2) a uniform language can be used for specification and requirements definition; and (3) requirements can be represented graphically. Most importantly, both basic and policy nondeterministic machines can function as *reusable specifications* that can be particularized by higher-level machines.

To formalize the idea of a policy HMS machine, we begin by introducing a new type of transition that can be used to define requirements for a lower-level machine.

Definition 11 (Policy Transition): δ is a "policy transition" over the set of states S if δ is of the form

id: (PRIMARIES) (B: $CONTROLS_1$, M: $CONTROLS_2$, E: $CONTROLS_3$) -->> (CONSEQUENTS)

where (1) id is a "label," (2) PRIMARIES is a (possibly empty) subset of S, (3) $CONTROLS_1$, $CONTROLS_2$ and $CONTROLS_3$ are a (possibly empty) sets of controls over S (respectively, called "beginning controls," "middle controls," and "end controls"), and (4) CONSEQUENTS is a (possibly empty) subset of S. The five sets in items (2)-(4) will be denoted $PRIMS(\delta)$, $CTRLS_B(\delta)$, $CTRLS_M(\delta)$, $CTRLS_E(\delta)$ and $CNSQS(\delta)$.

A policy HMS machine, with respect to some other HMS machine, can now be defined.

Definition 12 (Policy Machine): Given an HMS machine $H = (S, \Gamma_D, \Gamma_N)$, we say that a triple $H' = (S', \Gamma_D', \Gamma_N')$ is a "policy machine" for H if S' is a subset of S, and if Γ_D' and Γ_N' are (possibly empty) sets of policy transitions on S'. Γ_D' and Γ_N' will be called the "deterministic" and "nondeterministic" policy transitions of H, respectively.

This definition applies even if H is a policy machine itself. Then, an "n-level (multi-level) HMS specification" of a system is obtained by defining an ordered list $(H_1, \ldots, H_n)$ of machines such that H_1 is a basic HMS machine and each H_i, for i=2, ..., n, is a policy machine for H_{i-1}.

We now turn to the definition of the execution of such a list of HMS machines. Intuitively, a policy transition defines the desirable behavior of the next lower-level machine. While transitions in a basic HMS machine are instantaneous, a policy transition can take any number of time steps. A policy transition can begin if its beginning controls are satisfied and its primaries are true; it can continue if its middle controls are satisfied; it can complete when the consequents are true one moment after the end controls were true.

Thus, policy transitions *direct* transitions at lower levels by defining the *goals* or dynamic requirements. The achievement of a goal may involve numerous lower-level transitions that will have to be found by a search of the nondeterministic transitions at the lower level. This search can be automated to a large extent, and so the entire n-level specification becomes highly simplified and yet is itself executable.

We now introduce definitions leading to the formalization of policy machine execution.

Definition 13 (Policy Transition Enablement): Let H be a policy machine with marking M. Then a policy transition δ of H is "enabled" if (1) M(s, 0) = T for all s in PRIMS(δ), and (2) c is satisfied by M for all c in $CNTRL_B(\delta)$. The definitions of sets D-ENAB(H, M) and N-ENAB(H, M) from Section 2 extend to policy transition sets in a straightforward manner.

Definition 14 (Policy Firing Set): Let $H = (S, \Gamma_D, \Gamma_N)$ be a policy machine. Then Γ is a "policy firing set" of H from M if Γ = D-ENAB(H, M) $\cup \Gamma'$, where $\Gamma' \subseteq$ N-ENAB(H, M).

A policy firing set determines the set of policy transitions that *begin* to fire (or become active) at the higher level. We also need to define conditions under which higher-level transitions can *continue* and the conditions under which they can *terminate* (cease to be active).

Definition 15 (Policy Transition Continuation): A policy transition δ of a policy machine H can "continue" for marking M if, for all c in $CTRLS_M(\delta)$, c is satisfied by the marking M.

Definition 16 (Policy Transition Termination): A policy transition δ can "terminate" for marking M if $M \vdash CNSQS(\delta)$ and $M' \vdash CTRLS_E(\delta)$, where M'(s, i) = M(s, i-1) is the marking at the previous moment of time.

Using the definitions of transition enablement, continuation and termination, we can formalize the concept of a consistent execution path, or "plan," for any HMS machine. We begin with the syntactic definitions of a plan.

Definition 17 (Plan for Basic Machine): For a basic HMS machine H and initial marking M_0, $p = [\Gamma_0, \Gamma_1, \ldots]$ is a "plan for H from M_0" if p is a sequence of sets of transitions of H. The plan p is "internally consistent from M_0" if $[M_0, M_0[\Gamma_0], (M_0[\Gamma_0])[\Gamma_1], \ldots]$ is an execution of H from M_0. A finite prefix of a plan is called a "finite plan," and is internally consistent if it is the prefix of some internally consistent plan.

If p is an internally consistent plan for basic machine H from M_0, then $[M_0, M_0[\Gamma_0], (M_0[\Gamma_0])[\Gamma_1], \ldots]$ is called the "induced execution of H from M_0 via p." If $p = [\Gamma_0, \Gamma_1, \ldots, \Gamma_n]$ is an internally consistent finite plan, then we write $M_0[p]$ for the marking $(((M_0[\Gamma_0])[\Gamma_1]) \ldots)[\Gamma_n]$.

Definition 18 (Plan for Policy Machine): For a policy machine H, $p = [(\Gamma_0, \mathcal{E}_0, \mathcal{T}_0), (\Gamma_1, \mathcal{E}_1, \mathcal{T}_1), \ldots]$ is a "plan for H" if p is a sequence of triples of sets of policy transitions of H. The sets Γ_i, $\mathcal{E}_i$ and $\mathcal{T}_i$ are, respectively, the "firing set," "continuation set" and "termination set" of p at time i. The plan p "internally consistent" if it satisfies five conditions: (1) $\mathcal{E}_0$ and $\mathcal{T}_0$ are empty; (2) for all i, Γ_i and $\mathcal{E}_i$ are disjoint; (3) for all i, $\mathcal{T}_i \subseteq \Gamma_{i-1} + \mathcal{E}_{i-1}$; (4) for all i, $\mathcal{E}_i = \Gamma_{i-1} + \mathcal{E}_{i-1} - \mathcal{T}_i$; and (5) for all i, if $\delta \in \Gamma_i$ then there is a $j > i$ such that $\delta \in \mathcal{T}_j$. A finite prefix of a plan is called a "finite plan," and a finite plan is internally consistent if it is the prefix of some internally consistent plan.

The five conditions of internal consistency for a policy machine plan have intuitive meanings: (1) at the start of the plan, no transitions are already active; (2) no transition can fire

while it is continuing; (3) a transition can terminate only after it has begun firing; (4) all transitions that have begun firing either continue or terminate; and (5) every transition that begins firing must eventually terminate.

We next define what it means for a plan to be "consistent." Intuitively, a plan is consistent with a sequence of markings if the behavior suggested by the plan is compatible with the behavior suggested by the sequence of markings.

Definition 19 (Consistency of Plan for Basic Machine): Let $p = [\Gamma_0, \Gamma_1, \ldots, \Gamma_n]$ be a plan for basic machine H, and let $E = [M_0, M_1, \ldots]$ be a sequence of markings of the states of H. Then p is "consistent" with E if $[M_0, M_0[\Gamma_0], (M_0[\Gamma_0])[\Gamma_1], \ldots] = E$. A finite plan is consistent if it is the prefix of a consistent plan.

Definition 20 (Consistency of Plan for Policy Machine): Let $p = [(\Gamma_0, \mathcal{C}_0, \mathcal{T}_0), (\Gamma_1, \mathcal{C}_1, \mathcal{T}_1), \ldots]$ be a plan for policy machine H, and let $E = [M_0, M_1, \ldots]$ be a sequence of markings of the states of H. Then p is "consistent" with E if, for all i, (1) Γ_i is a firing set of H from M_i; (2) every transition in $\mathcal{C}_i$ can continue from M_i; and (3) every transition in $\mathcal{T}_i$ can terminate from M_i.

Only one more concept, that of an "induced execution from a plan," is needed before the execution of an n-level specification can be given. We saw in Definition 17 that if a plan $[\Gamma_0, \Gamma_1, \ldots, \Gamma_n]$ for a basic HMS machine is internally consistent from M_0, then it induces the sequence of markings $[M_0, M_0[\Gamma_0], (M_0[\Gamma_0])[\Gamma_1], \ldots]$. The same is true for a plan of a policy machine: if the plan is internally consistent, then it induces a sequence of markings on its states from a single marking.

Definition 21 (Induced Execution from Policy Plan): Let $p = [(\Gamma_0, \mathcal{C}_0, \mathcal{T}_0), (\Gamma_1, \mathcal{C}_1, \mathcal{T}_1), \ldots]$ be a plan for policy machine H, and let M_0 be a marking for H. Then $E = [M_0, M_1, \ldots]$ is the "induced execution of H from M_0 via p" if, for every $i > 0$,

(1) $M_i(s, 0) = T$ for every state in $CNSQS(\mathcal{T}_i)$;
(2) $M_i(s, 0) = F$ for every state in $PRIMS(\Gamma_i) - CNSQS(\mathcal{T}_i)$;
(3) $M_i(s, 0) = M_{i-1}(s, 0)$ for every state in neither $PRIMS(\Gamma_i)$ nor $CNSQS(\mathcal{T}_i)$; and
(4) $M_i(s, j) = M_{i-1}(s, j+1)$ for every state, and for every $j < 0$.

Notice the similarity between the induced execution of a policy machine (Def. 21) and the execution of a basic HMS machine (Defs. 9-10). The execution of an n-level specification can now be given. For this definition, we assume that a plan $[\Gamma_0, \Gamma_1, \ldots]$ for a basic machine is written as if it were a plan for a policy machine: $[(\Gamma_0, \{\}, \{\}), (\Gamma_1, \{\}, \Gamma_0), (\Gamma_2, \{\}, \Gamma_1), \ldots]$.

Definition 22 (n-Level Specification Execution): Let $\mathfrak{H} = (H_1, \ldots, H_n)$ be an n-level specification with initial marking M. Then an "execution of $\mathfrak{H}$ from M" is a list $(p_1, \ldots, p_n)$, where $p_i = [(\Gamma_{i0}, \mathcal{C}_{i0}, \mathcal{T}_{i0}), (\Gamma_{i1}, \mathcal{C}_{i1}, \mathcal{T}_{i1}), \ldots]$ is an internally consistent plan for H_i, such that (1) for all $i \geq 1$ and all $j \leq i$, p_i is consistent with the induced execution of H_j from M via p_j; and (2) for all $i < n$, and for all $j \geq 0$, $PRIMS(\Gamma_{i+1,j}) \subseteq PRIMS(\Gamma_{ij})$ and $CNSQS(\mathcal{T}_{i+1,j}) \subseteq CNSQS(\mathcal{T}_{ij})$.

These two conditions have the following intuitive meanings: (1) the plan at every level meets the requirements imposed by all higher levels; and (2) the plan at every level is implemented by the plan at the next lower level.

Notice that the execution of an n-level specification $\mathfrak{H} = (H_1, ..., H_n)$ from marking M_0 can begin with a plan for the highest machine H_n, which needs only to be consistent at its own level. Then, dropping down one level, a plan can be found for H_{n-1} which is consistent at its own level, and which is a consistent implementation of the first plan. Continuing in this manner, the execution can be found one level at a time, with higher-level plans providing hints for, and imposing requirements upon, plans at lower-levels.

As an illustration of concept of multi-level HMS specification, consider the pair (H_1, H_2) in Figure 3.1, together with the initial marking M_0, where $M_0(A, 0) = T$, $M_0(s, i) = F$ otherwise.

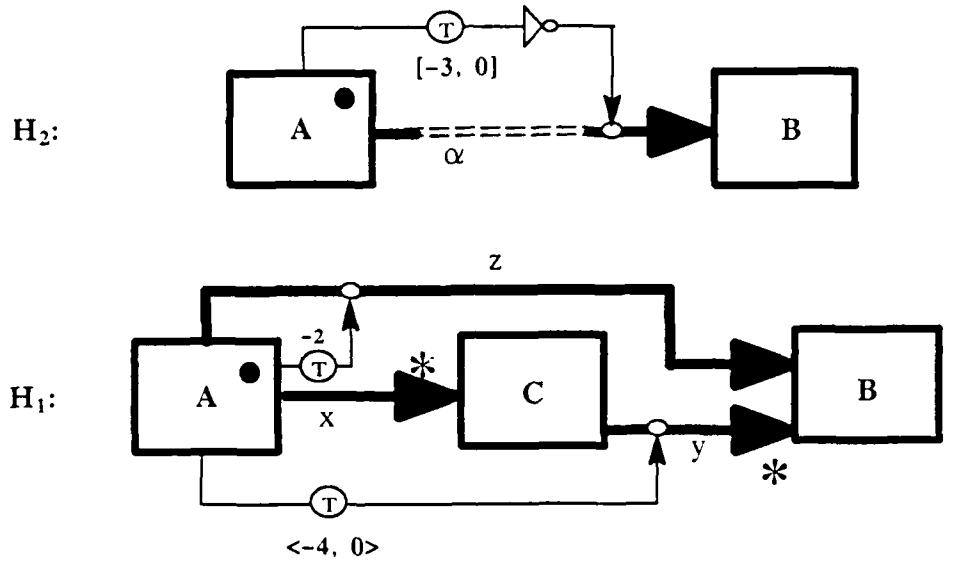


Figure 3.1. Example of a 2-Level HMS Specification

The following is a plan p_2 for the policy machine H_2:

p_2: $[(\{\alpha\}, \{\}, \{\}), (\{\}, \{\alpha\}, \{\}), (\{\}, \{\alpha\}, \{\}), (\{\}, \{\alpha\}, \{\}), (\{\}, \{\}, \{\alpha\}), (\{\}, \{\}, \{\}), ...]$.

In this plan, policy transition α fires at the first moment and terminates four moments later. The induced execution of H_2 from M_0 via p_2 is $E = [M_0, M_1, M_2, M_3, M_4, M_5, ...]$, where

$M_1(A, -1) = T$, and $M_1(A, i) = M_1(B, i) = F$ otherwise.
$M_2(A, -2) = T$, and $M_2(A, i) = M_2(B, i) = F$ otherwise.
$M_3(A, -3) = T$, and $M_3(A, i) = M_3(B, i) = F$ otherwise.
$M_4(A, -4) = T$, $M_4(B, 0) = T$, and $M_4(A, i) = M_4(B, i) = F$ otherwise.
$M_5(A, -5) = T$, $M_5(B, 0) = M_5(B, -1) = T$, and $M_5(A, i) = M_5(B, i) = F$ otherwise, etc.

Notice that the plan p_2 is consistent with E (by Def. 20): A is initially marked and it has been unmarked for three moments when α completes. Here are some possible plans for H_1:

p_1: [{x}, { }, { }, { }, {y}, { }, ...]
p'_1: [{x}, { }, { }, {y}, { }, ...]
p''_1: [{ }, { }, {z}, { }, ...]

Although all three of these plans are consistent with the marking M_0, only p_1 is consistent with the induced execution E. In this case, the induced execution of H_1 from M_0 via p_1 is $[M_0, M_0[\{x\}], M_0[\{x\}, \{\}], ...]$, and $M_0[\{x\}]$ agrees with M_1 on states {A, B}, $M_0[\{x\}, \{\}]$ agrees with M_2 on {A, B}, and so forth.

Thus, (p_1, p_2) is a consistent execution of n-level specification $\mathfrak{H}$ from initial marking M_0.

4. Verification of Consistency of Multi-Level HMS Specifications

We now address a restricted method of verifying that a multi-level HMS specification $\mathfrak{H} = (H_1, ..., H_n)$, together with initial conditions, is consistent. It is sufficient to demonstrate that there exists an execution (Def. 22) of $\mathfrak{H}$ from any marking M_0 satisfying the initial conditions. In this section, we present a method for verifying that such an execution exists, given a partially defined and possibly misordered plan for the lowest-level machine H_1. A practical example of this will be presented in the next section.

The application of the verification method of this section postulates the existence of a "potential plan" for accomplishing higher-level deterministic transitions that define requirements. A potential plan is a sequence of concurrent sets of transitions at the lowest-level basic HMS machine that can lead to the satisfaction of the requirement, *if there were no constraints*. Our method can then determine mechanically whether there is *any* scheduling of these transitions (involving delays and possible reorderings) that will satisfy the constraints. In addition, since the method is constructive, correct schedules can be derived if they exist.

It is useful to contrast our verification technique, which addresses "liveness" issues, with methods that address "safety" issues [La-77]. A plan can be considered as a possible means of achieving a liveness property. The method to be presented in this section determines whether a potential plan can be properly scheduled (proving liveness), but is not designed to discover that no feasible plan exists (disproving liveness). In [Fr-89], we present a safety verification method which establishes the unreachability of "safety states" using correctness-preserving transformations on HMS machines (proving safety), but is not designed to show reachability of safety states (disproving safety).

The restricted form of the verification method that we consider is the following: (1) We assume that we wish to determine if a *single* high-level deterministic policy transition in an n-level HMS specification is achievable at the lowest-level basic machine. (2) We assume that we are given (or we can discover) a potential plan (sequence of sets of transitions) that contains all the necessary steps for reaching the goal. Such a plan can often be derived by ignoring all the controls and by attempting to go from the primary states of the deterministic policy transition to its consequents. (3) We wish to determine if there is *a feasible schedule*

for firing the nondeterministic transitions in the plan that will achieve the consequents of the high-level. (4) If there are such schedules, we wish to identify them.

As an example, suppose that we have a potential plan {x, y} z w {x, w} y , where each letter corresponds to a nondeterministic transition and concurrent firings are indicated in braces. In this plan, x and y are fired in the first moment, then z is fired, then w is fired, an so on. To find a feasible schedule, we must derive the correct ordering and intermediate delays that will make the plan feasible. This can be expressed parametrically using the following definition, where ϕ represents the empty firing set, indicating a *wait* (no action).

Definition 23 (Variable Delay Plan): Let p_0 be a plan for a basic HMS machine. Then p is a "variable delay plan" for p_0 if p is obtained from p_0 by the inclusion of a number of terms of the form ϕ^n, where the exponent indicates the number of steps waited. Each ϕ^n is called a "variable delay."

For the example being considered, one variable delay plan is ϕ^i {x, y} ϕ^j z w ϕ^k {x, w} ϕ^n y. Interestingly, it turns out that allowing exponents to be negative, results in the discovery of reordering of the steps in the plan. Our goal is to determine mathematically the feasible solutions to such exponents that will make the variable plan feasible or to determine that no such solution is possible. This can be a non-trivial task since the HMS machines may have complex logical and temporal controls on transitions.

Overview of Solution Technique for Variable Delay Plans: Given a variable delay plan p and an initial marking M_0, our verification techniques involves a three-stage process:

A. Apply a set of "postconditions laws" to obtain symbolic facts that will be true after each step of of the plan is executed.

B. Apply a set of "precondition laws" that determine what facts must be true before each step of the plan *can* be executed.

C. Use the results of A and B to obtain a set of numerical inequalities in terms of the variable delays of p. Either demonstrate that the system of inequalities is inconsistent or obtain a solution that allows the replacement of variable delays in p with constant delays.

We now present the postcondition and precondition laws and we provide a sketch of how inequalities for solving for the variable delays can be obtained. Recalling the definitions of Section 2, $M \vdash C$ indicates that the set of controls C (interpreted as the conjunction of the individual controls in C) is true in the marking M.

A. Postcondition Laws. The following four types of postcondition laws determine symbolically what facts can be derived as the plan is executed (Δ is a transition set in p):

INITIAL:

If $M \vdash C$ and c is a conjunct of C, then $M \vdash c$

FIRING:

If $s \in CNSQS(\Delta)$, then $M[\Delta] \vdash (s, [0, 0])$

If $s \in PRIMS(\Delta)$ and $s \notin CNSQS(\Delta)$, then $M[\Delta] \vdash (\neg s, [0, 0])$

DRIFT:

If $M \vdash (x, [\alpha, \beta])$, $\beta < 0$, then $M[\phi^i] \vdash (x, [\alpha - i, \beta - i])$

If $M \vdash (x, [\alpha, \beta])$, $\beta < 0$ then $M[\Delta] \vdash (x, [\alpha - 1, \beta - 1])$

If $M \vdash (s, [\alpha, 0])$, $s \in PRIMS(\Delta)$, $s \notin CNSQS(\Delta)$, then $M[\Delta] \vdash (s, [\alpha - 1, -1])$

If $M \vdash (\neg s, [\alpha, 0])$, and $s \in CNSQS(\Delta)$, then $M[\Delta] \vdash (s, [\alpha - 1, -1])$

STRETCH:

If $M \vdash (x, [\alpha, 0])$, then $M[\phi^i] \vdash (x, [\alpha - i, 0])$

If $M \vdash (x, [\alpha, 0])$, and $x \notin PRIMS(\Delta) \cup CNSQS(\Delta)$, then $M[\Delta] \vdash (x, [\alpha-1, 0])$

B. **Precondition Laws.** The following four types of precondition laws determine symbolically what facts need to be true as the plan is executed (Δ is a transition set in p):

FINAL:

If $M \vdash C$ needs to be true at the end of p and c is a conjunct of C, then $M[p] \vdash c$.

PRIMARY:

If $p = p_1 \Delta p_2$, and $s \in PRIMS(\Delta)$, then $M[p_1] \vdash (s, [0, 0])$

CONTROL:

If $p = p_1 \Delta p_2$, and $c \in CTRLS(\Delta)$, then $M[p_1] \vdash c$

UPPER:

If $p = p_1 p_2$, $c \in BEGIN\text{-}CTRLS(\Delta)$, $|p_1| = n$, and $\Delta \in \Gamma_n$, then $M[p_1] \vdash c$

If $p = p_1 p_2$, $c \in MID\text{-}CTRLS(\Delta)$, $|p_1| = n$, and $\Delta \in \mathcal{C}_n$, then $M[p_1] \vdash c$

If $p = p_1 p_2$, $c \in END\text{-}CTRLS(\Delta)$, $|p_1| = n$, and $\Delta \in \mathcal{T}_{n+1}$, then $M[p_1] \vdash c$

C. **Derivation and Solution of Inequalities:** After the application of steps A and B, for each initial subplan of p', we compare the facts $M[p'] \vdash C_1$ derived from A to the necessary facts $M[p'] \vdash C_2$ derived from B. Then, for each control (x, τ') in C_2, where x is s or $\neg s$, there must exist an (x, τ) in C_1 such that

- If $\tau' = [\alpha', \beta']$ and $\tau = [\alpha, \beta]$, then $[\alpha, \beta] \subseteq [\alpha', \beta']$, resulting in the inequalities $\alpha \geq \alpha'$, $\beta \leq \beta'$.
- If $\tau' = \langle\alpha', \beta'\rangle$ and $\tau = [\alpha, \beta]$, then $\langle\alpha', \beta'\rangle \in [\alpha, \beta]$, resulting in the inequalities $\beta \geq \alpha' \geq \alpha$ or $\beta \geq \beta' \geq \alpha$.

[Note: We assume that no fact uses a sometime-change-delay, since any control of the form $(x, \langle\alpha, \beta\rangle!)$ can be replaced by the conjunction of $(\neg x, \alpha-1)$ and $(x, \langle\alpha, \beta\rangle)$.]

If, for some (x, τ') in C_2, no control in C_1 is of the form (x, τ), then the original plan is inadequate. The set of all inequalities can be solved using standard mathematical techniques to obtain the exact values of delays in the plan p that can realize the desired policy transition at a higher-level policy HMS machine. On the other hand, if there are no solutions, it again follows that the original plan was inadequate.

As indicated earlier, by allowing exponents in the solution to the system of inequalities to be negative, reordering of the sets of transitions can be obtained. A negative exponent causes the remainder of the plan to be shifted back in time by the magnitude of the delay.

This completes the sketch of our restricted method for verifying that an n-level specification is consistent. To review, this method determines a scheduling and reordering of a potential plan that can satisfy a set of high-level requirements in an n-level specification.

5. Example: Engine Monitoring System

In this section, we will illustrate the multi-level specification and verification concepts of this paper on a realistic example. Consider an embedded software system that monitors a steam generator for a pressurized water reactor (PWR). Temperature and pressure are independently and periodically sensed, numerically calculated from sensor readings, and printed. When both temperature and pressure are known the volume can be numerically derived and printed. The sensor control, numerical calculations and printer control are handled by software modules for which lower timing bounds are given. The HMS machine H_1 of Figure 5.1 provides a specification of this system, where a vertical bar represents the empty primary set for a transition. Recall that an asterisk next to a transition arrow indicates nondeterminism. Thus, H_1 may be considered as a generic specification of a whole class of PWR systems, with the specific behavior left undetermined while the possible *choice* of actions is specified precisely.

We can add requirements to H_1 by controlling its execution with the policy machine H_2 of Figure 5.2. In addition to providing hints about the overall flow of activity at the lower level, H_2 imposes three types of restrictions on the monitoring system: (1) temperature cannot be printed while or soon after pressure is printed, and vice versa; (2) the calculation of volume should not be undertaken while either of the two normal sensing cycles is underway; and (3) the calculation of volume should only be undertaken if it is based on recent temperature and pressure readings. Note that all the policy transitions in H_2 are nondeterministic.

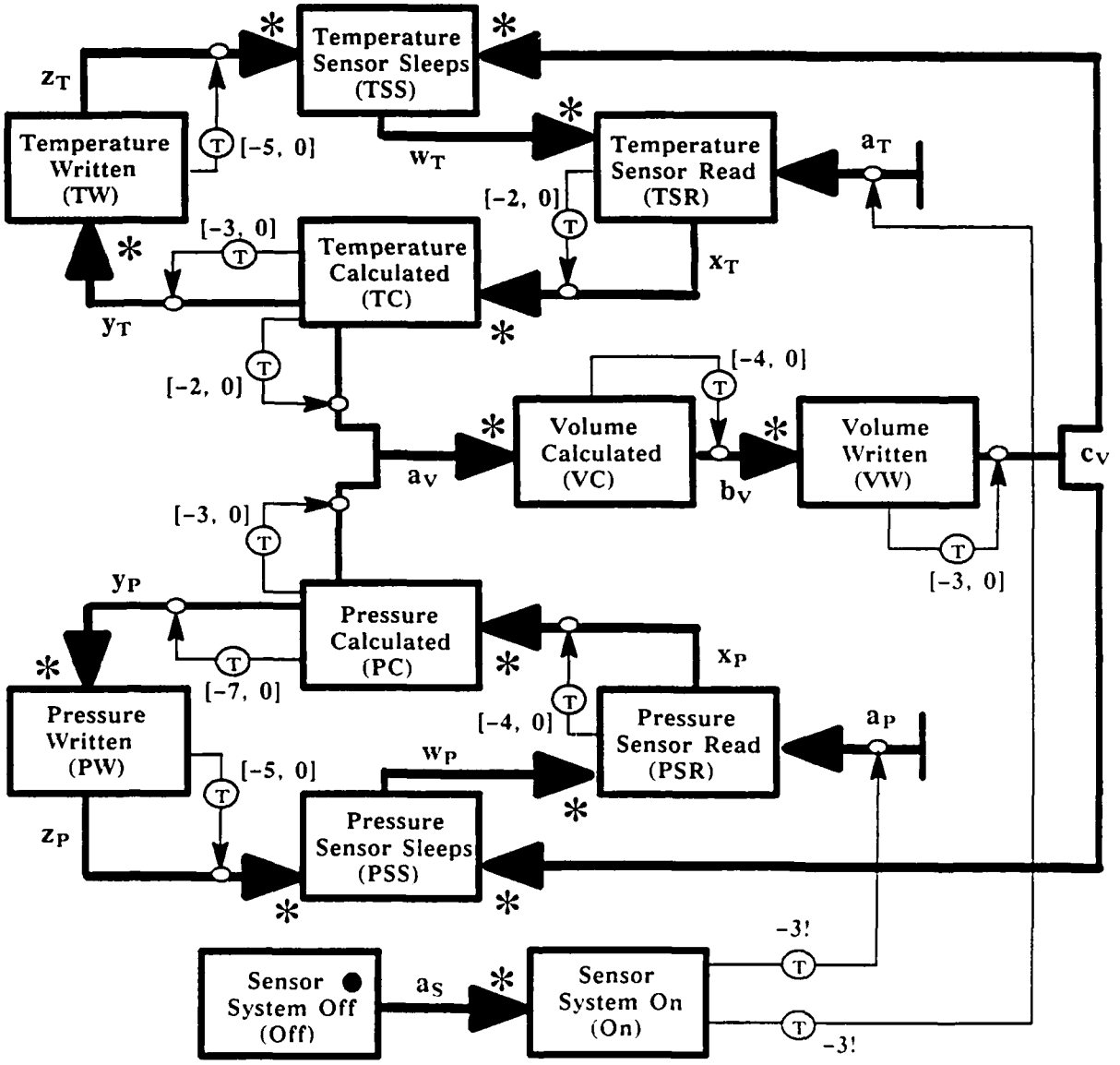


Figure 5.1. PWR Steam Generator Monitoring System: Low-Level Specification H_1

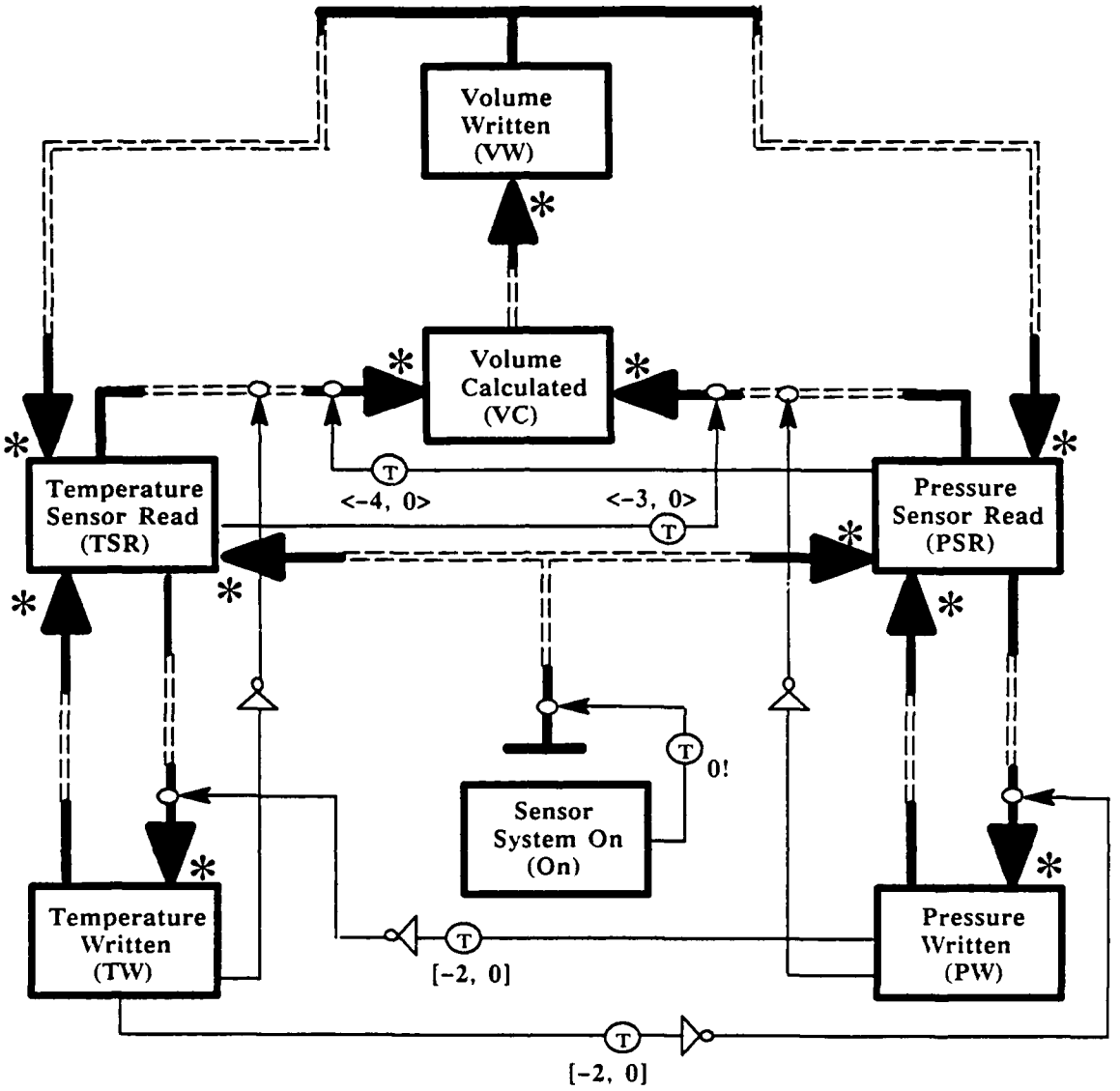


Figure 5.2. PWR Steam Generator Monitoring System: Lower Requirements H_2

There is yet another level of requirements for this monitoring system, which can be specified by a policy machine H_3. This machine controls the policy machine H_2, imposing three additional requirements on the system: (1) there is a fixed time interval between temperature readings; (2) there is a fixed time interval between pressure readings; and (3) there is a fixed deadline by which the volume must be written.

Notice that all the transitions in H_3 are deterministic. Thus, the execution of the 3-level specification (H_1, H_2, H_3) requires that all transitions in H_3 be fired whenever they are enabled. This can be achieved at the lower levels only if there is an infinite plan such that (1) Temperature Sensor Read is entered every 22 moments; (2) Pressure Sensor Read is entered every 25 moments; and (3) Volume Written is entered at most 50 moments after

Sensor System On first becomes true. The specification is inconsistent if there is no such infinite plan.

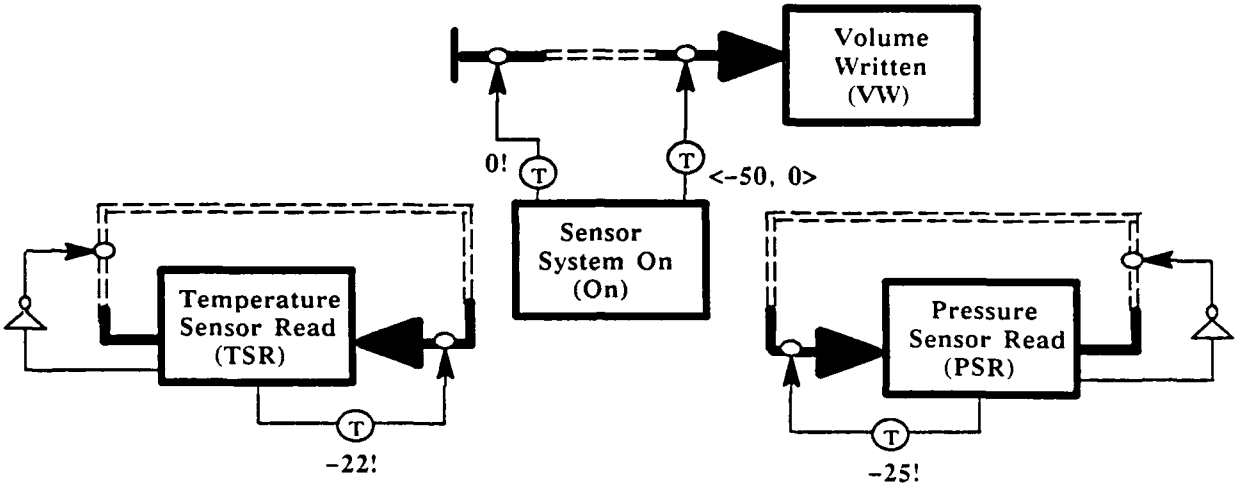


Figure 5.3. PWR Steam Generator Monitoring System: Upper Requirements H_3

To illustrate the proof method described in Section 4, we will demonstrate a claim which is a part of the overall consistency requirement: there is a finite plan p such that (1) Temperature Sensor Read is entered every 22 moments, and no more than 22 moments separate the last entrance from the end of the finite plan; (2) Pressure Sensor Read is entered every 25 moments, and no more than 25 moments separate the last entrance from the end of the finite plan; and (3) Volume Written is entered at most 50 moments after Sensor System On first becomes true.

We begin with a formal presentation of the initial marking M and the requirements on the final marking M[p] after the plan p is executed:

$M \vdash (Off, [-\infty, 0])\ (\neg On, [-\infty, 0])\ (\neg TSS, [-\infty, 0])\ (\neg TSR, [-\infty, 0])\ \ldots\ (\neg VW, [-\infty, 0])$

$M[p] \vdash (TSS, 0)\ (PSS, 0)\ (VW, -1)\ (TSR, <-22, 0>!)\ (PSR, <-25, 0>!)$

If a finite plan p can be found that satisfies these requirements, then the claim will have been constructively demonstrated.

We first guess that the volume can be calculated and written after one loop around each of the pressure and temperature cycles. Next, we guess an ordering of these loops to get the following finite plan p':

$$a_S \{a_T, a_P\}\ x_T\ y_T\ x_P\ z_T\ y_P\ z_P\ w_T\ w_P\ x_T\ x_P\ a_V\ b_V\ c_V$$

We are not concerned about possible misorderings of actions in this plan, since the proof method can discover and correct such errors. We also note that the policy machine H_2 can be used as an intermediate step in the construction of a candidate plan. In this case, a plan

which visits the states Temperature Written and Pressure Written once each can be guessed at this level, and the plan p' can then be guessed as a solution to this intermediate plan.

Next, we extend p' into a variable delay finite plan p by interleaving variable pauses between actions. For simplicity, we omit pauses at the start and end of p', since these are known to be unnecessary:

$$p = a_S\, \phi^{i1}\, \{a_T, a_P\}\, \phi^{i2}\, x_T\, \phi^{i3}\, y_T\, \phi^{i4}\, x_P\, \phi^{i5}\, z_T\, \phi^{i6}\, y_P\, \phi^{i7}$$
$$z_P\, \phi^{i8}\, w_T\, \phi^{i9}\, w_P\, \phi^{i10}\, x_T\, \phi^{i11}\, x_P\, \phi^{i12}\, a_V\, \phi^{i13}\, b_V\, \phi^{i14}\, c_V.$$

The three steps of the proof method are now sketched for this example.

A. Generate postconditions of all prefixes of p, using the postcondition laws. Only a few of these facts are shown below:

$M \vdash (\neg On, [-\infty, 0])$ by Initial Law

$M[a_S] \vdash (\neg On, [-\infty, -1])\ (On, 0)$ by Drift Law, Primary Law

$M[a_S\, \phi^{i1}] \vdash (\neg On, [-\infty, -i1-1])\ (On, [-i1, 0])$ by Drift Law, Stretch Law.

...

$M[a_S\, \phi^{i1}\, \{a_T, a_P\}\, \phi^{i2}\, x_T\, \phi^{i3}\, y_T] \vdash (PSR, [-i3-i2-2, 0])$

$\mathbf{M[a_S\, \phi^{i1}\, \{a_T, a_P\}\, \phi^{i2}\, x_T\, \phi^{i3}\, y_T\, \phi^{i4}] \vdash (PSR, [-i4-i3-i2-2, 0])}$ by Drift Law.

...

$M[a_S\, \phi^{i1}\, \{a_T, a_P\}\, \phi^{i2} \ldots \phi^{i11}\, x_P] \vdash (TSR, [-i11-i10-i9-3, -i11-2])$

$\mathbf{M[a_S\, \phi^{i1}\, \{a_T, a_P\} \ldots \phi^{i11}\, x_P\, \phi^{i12}] \vdash (TSR, [-i12-i11-i10-i9-3, -i12-i11-2])}$ by Drift Law

...

B. Generate preconditions of all prefixes of p that end with a variable pause. Only a few of these facts are shown below:

$\mathbf{M[a_S\, \phi^{i1}\, \{a_T, a_P\}\, \phi^{i2}\, x_T\, \phi^{i3}\, y_T\, \phi^{i4}] \vdash (PSR, [-4, 0])}$ by Control Law

$\mathbf{M[a_S\, \phi^{i1}\, \{a_T, a_P\} \ldots \phi^{i11}\, x_P\, \phi^{i12}] \vdash (TSR, <-3, 0>)}$ by Upper Law

$M[a_S\, \phi^{i1}\, \{a_T, a_P\}\, \phi^{i2} \cdots \phi^{i14}\, c_V] \vdash (TSS, 0) \ldots (PSR, <-25, 0>!)$ by Final Law

C. Derive constraints and associated inequalities using the results of the previous two steps. For example, from the four boldfaced facts above we can derive the following:

$[-4, 0] \subseteq [-i4-i3-i2-2, 0]$ implies $i4 + i3 + i2 + 1 \geq 4$

$<-3, 0> \in [-i12-i11-i10-i9-3, -i12-i11-2]$ implies $i12 + i11 + 2 \leq 3$.

The full set of inequalities is as follows:

$i1 = 3 \quad i2 \geq 2 \quad i3 \geq 3 \quad i4 + i3 + i2 + 1 \geq 4 \quad i5 + i4 + 1 \geq 5$

$i6 + i5 + 1 \geq 7 \quad i6 \geq 2 \quad i7 \geq 5 \quad i10 + i9 + 1 \geq 2 \quad i11 + i10 + 1 \geq 4$

$i8 + i7 + i6 + i5 + i4 + i3 + i2 + 6 = 22 \quad i9 + i8 + i7 + i6 + i5 + i4 + i3 + i2 + 6 = 25$

$i12 + i11 + 2 \leq 3 \quad i12 + 1 \leq 4 \quad i12 + i11 + 1 \geq 2 \quad i12 \geq 3 \quad i13 \geq 4$

$i14 \geq 3 \quad i14 + i13 + i12 + i11 + 4 \leq 22 \quad i14 + i13 + i12 + 3 \leq 25$

$i14 + i13 + i12 + i11 + i10 + i9 + 7 \geq 23 \quad i14 + i13 + i12 + i11 + i10 + 6 \geq 26.$

This set of inequalities has the following solution:

$$i1 = 3 \quad i2 = 2 \quad i3 = 3 \quad i4 = 0 \quad i5 = 4 \quad i6 = 2 \quad i7 = 5$$
$$i8 = 0 \quad i9 = 3 \quad i10 = 5 \quad i11 = -2 \quad i12 = 3 \quad i13 = 4 \quad i14 = 3.$$

From these values, we obtain the following finite plan:

$a_S \phi^3 \{a_T, a_P\} \phi^2 x_T \phi^3 y_T x_P \phi^4 z_T \phi^2 y_P \phi^5 z_P w_T \phi^3 w_P \phi^5 x_T \underline{\phi^{-2}} x_P \phi^3 a_V \phi^4 b_V \phi^3 c_V$

The elimination of the one underlined delay with a negative exponent is achieved by shifting back the subplan after the "ϕ^{-2}" term by two steps and recombining terms:

$a_S \phi^3 \{a_T, a_P\} \phi^2 x_T \phi^3 y_T x_P \phi^4 z_T \phi^2 y_P \phi^5 z_P w_T \phi^3 w_P \phi^5 x_T \phi^{-2} x_P \phi^3 a_V \phi^4 b_V \phi^3 c_V =$

$a_S \phi^3 \{a_T, a_P\} \phi^2 x_T \phi^3 y_T x_P \phi^4 z_T \phi^2 y_P \phi^5 z_P w_T \phi^3 w_P \phi^4 \phi x_T$

$x_P \phi \phi^2 a_V \phi^4 b_V \phi^3 c_V =$

$a_S \phi^3 \{a_T, a_P\} \phi^2 x_T \phi^3 y_T x_P \phi^4 z_T \phi^2 y_P \phi^5 z_P w_T \phi^3 w_P \phi^4 x_P x_T \phi^2 a_V \phi^4 b_V \phi^3 c_V$

This plan for the lowest-level machine begins by turning on the system (a_S), and then waits three moments (ϕ^3), starts reading both temperature and pressure ($\{a_T, a_P\}$), waits two moments (ϕ^2), starts calculating temperature (x_T), waits three moments (ϕ^3), starts writing temperature (y_T), starts calculating pressure (x_P), waits four moments (ϕ^4), and so forth.

Notice that two other solutions to the inequalities above can be found by varying only the values of i4 and i5: (a) i4 = –1 and i5 = 5, or (b) i4 = –2 and i5 = 6. These solutions would further reorder the original plan by (a) making the first temperature write and first pressure calculation simultaneous, or (b) making the first pressure calculation immediately precede the first temperature write.

6. Summary

In this paper, a new approach was introduced for multi-level specification of real-time software in terms of HMS abstract machines. The major benefits of the approach are (1) significant reduction in complexity of specifications since lower-level details can be ignored as requirements are defined, and (2) reusability of nondeterministic specifications through the use of higher-level policy machines that impose requirements on them. In addition, a restricted method of verifying the consistency of multi-level specifications was presented. Because of the constructive nature of this method, given the *set* of tasks necessary to satisfy a requirement, the correct *ordering* and intertask *delays* can be derived algorithmically. The utility of the verification method was demonstrated for a three-level specification of an embedded software system by deriving a task execution schedule that satisfies a set of high-level requirements.

References

[Ab-88] Abadi, M. and L. Lamport, "The existence of refinement mappings," *Proc. of the Logic in Computer Science Conference*, Edinburgh, Scotland, July 1988.

[Ch-81] Cheheyl, M.L., M. Gasser, G.A. Huff, and J.K. Millen, "Verifying security," *ACM Computing Surveys*, Vol. 13, No. 3, September 1981, pp. 279-339.

[Fr-89] Franklin, M.K., and A. Gabrielian, "A transformational method for verifying safety properties in real-time systems," submitted for publication.

[Ga-87] Gabrielian, A., and M. E. Stickney, "Hierarchical representation of causal knowledge," *Proc. WESTEX-87, IEEE Expert Systems Conf.*, June 1987, pp. 82-89.

[Ga-88a] Gabrielian, A., and M. K. Franklin, "State-based specification of complex real-time systems," *IEEE Real-Time Systems Symposium*, 1988, pp. 2-11.

[Ga-88b] Gabrielian, A., and M.K. Franklin, "Hierarchical Multi-State (HMS) specification of real-time systems," *Proceedings of the ONR Kickoff Workshop on the Foundations of Real-Time Computing Research Initiative*, Falls Church, VA, November 29-30, 1988, pp. 96-100. Reprinted in *Real-Time Systems Newsletter*, Vol. 5, No. 1, Winter 1989, pp. 8-12.

[Ge-86] Gehani, N., and A. McGettrick, *Software Specification Techniques*, Addison-Wesley, Wokingham, England, 1986.

[Ha-87] Harel, D., "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, Vol. 8, No. 3, June 1987, 231-274.

[Ja-88] Jahanian, F., R. Lee and A.K. Mok, "Semantics of Modecharts in Real Time Logic," 21st *Hawaii International Conference on System Science*, January 5-8, 1988, Kailua-Kona, Hawaii, pp. 479-489.

[La-77] Lamport, L., "Proving the correctness of multiple programs," *IEEE Trans. on Software Engineering*, Vol. SE-3, No. 2, March 1977, pp. 125-143.

[Re-82] Reisig, W., *Petri Nets, An Introduction*, Springer-Verlag, Berlin, 1982.

[St-88] Stankovic, J.A., and K. Ramamritham, *Hard Real-Time Systems (Tutorial)*, IEEE Computer Society Press, Wahington, D.C., 1988.